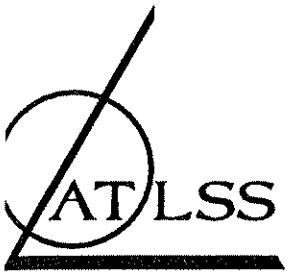
9-1-1994

Resource Constrained Project Scheduling Using Local Search in 'Problem Space'

InKyoung Park

S. David Wu

Robert H. Storer



ADVANCED TECHNOLOGY FOR
LARGE
STRUCTURAL SYSTEMS

Lehigh University

Resource Constrained Project Scheduling Using Local Search in 'Problem Space'

by

InKyoung Park
ATLSS Research Assistant

S. David Wu
Professor of Industrial Engineering

Robert H. Storer
Professor of Industrial Engineering

ATLSS Report No. 94-03

September 1994

ATLSS Engineering Research Center
Lehigh University
117 ATLSS Dr., Imbt Laboratories
Bethlehem, PA 18015-4729
(610) 758-3525

An NSF Sponsored Engineering Research Center

1. Introduction

Project management techniques such as the critical path method (CPM) and the precedence diagraming method (PDM) have become commonplace in construction management. While of value as management tools, they overlook important problem constraints. Most important among these is that fact that project management tools assume no conflict between resource requirements for various activities. When the resources are scarce from a schedule based on CPM, the activities have to wait for available resources. Then a main delay of the entire project is expected. Many authors (Jaafari, 1984; Chrzanowski and Johnston, 1986; Pultar, 1990) have suggested the disadvantages of CPM/PDM.

The Construction Industry Institute (1990) reported that one-third of construction industry projects experience cost overrun or schedule slippage. A construction project manager must consider such requirements as labor, material, time, cash flow, and weather. Architects and engineers alike must consider the planning, scheduling, and coordination of a large variety of construction activities. Project managers must consider the cost and time tradeoffs among various resources in a project schedule. Most cost functions in construction scheduling are either time based such as project duration and due dates, or cost based such as cash flow (e.g. net present value) and total cost.

While CPM/PDM consider time only, resource constraints are typically handled by project managers in an ad-hoc fashion. Construction projects are prototypical applications of "resource constrained project scheduling" (RCPS). RCPS is known to be a challenging combinatorial optimization problem (Tavares and Weglarz, 1990). RCPS with minimum makespan objective is a classical combinatorial optimization problem of great interest from both practical and theoretical standpoints. As a generalization of the job shop scheduling problem (JSP), RCPS is known to be NP-hard in a strong sense.

Various heuristic procedures for RCPS have been presented and compared by various researchers (Davis and Patterson, 1975; Boctor, 1990; Olaguibel and Goerlich, 1989). For small or moderate size problems (e.g., up to 50 activities) branch and bound schemes (Christofides et

al, 1987; Stinson et al, 1978) and A* search (Bell and Park, 1990) have been developed. Talbot and Patterson (1978) presented an integer programming algorithm and showed good results for small projects (30 activities or less). Since practical problems such as construction scheduling are typically much larger (Scherer and Rotman, 1992; Tsubakitani and Deckro, 1990), heuristic solution procedures become attractive. But the result of heuristic procedure itself is a little far from a near-optimal or optimal solution in a large-scale problem. To improve the heuristic solution, some local search methods can be used. However, applying a typical local search based on a pairwise swap neighborhood often wastes much of its time checking solution feasibility, and does not result in high quality solutions. An inherent weakness in such a search neighborhood is its failure to incorporate problem specific information which could help to identify promising candidate solutions.

Storer, Wu, and Vaccari (1992) suggested a "problem space search" based on a new search neighborhood which exploits problem specific knowledge. They have achieved some very encouraging results for the job shop scheduling problem. While a typical local search method examines the solution space for good solutions to a specific problem, a problem space search method examines the space of possible perturbations of the problem data. The problem space search combines the power of the specific tailored heuristic with local search. Alternatively, Li and Willis (1992) suggested an iterative method with a multi-pass scheduling procedure using forward and backward scheduling. Their iterative procedure needs to check solution feasibility constantly and can be easily stuck by local optimums.

In this paper, "problem space search" is implemented along with a generalized priority rule approach and an enhanced application of genetic algorithms and an evolutionary programming approach.

2. RCPS and Problem Space

2.1 The Resource Constrained Project Scheduling Problem

As mentioned earlier, CPM is ineffective in resource constrained scheduling problems. In RCPS an activity could need multiple resources during its processing. Each activity could have sets of predecessor and successor activities. Each resource has a limited capacity and availability. Especially the concept of disjunctive graph does not work well in RCPS. While in JSP each operation corresponds to a specific arc set disjunctive, in RCPS each activity corresponds to several disjunctive arc sets. An RCPS with a minimum makespan objective is much more difficult than its JSP counterpart. When all realistic constraints of limited resources are imposed in RCPS, it would be difficult to just check the feasibility of each solution considered. Consider an RCPS with the following assumptions: Once started, an activity must be completed (no preemption is allowed); and once committed to specific activities, a resource cannot be reassigned or have its resource level altered until the activity is complete. The objective is to complete the entire project in minimal time (makespan). The formulation due to Olaguibel and Goerlich (1989) is presented as follows:

$$\text{objective} \qquad \text{Min } \{ \text{Max } (t_i + d_i) \}$$

$$\text{subject to} \qquad t_j - t_i \geq d_i \quad (i,j) \in A,$$

$$\sum_{S(t)} r_{ik} \leq b_k, \; t = 1, \ldots, T; \; k = 1, \ldots, K$$

where t_i : starting time of activity $i = 1, \ldots, n$

A: set of precedence constraints

d_i : processing time of activity i

r_{ik} : number of resource k required by activity i

b_k : total availability of resource k

S(t): set of activities in process at time t

K: number of resource types

Any feasible solution of the above problem is a schedule for RCPS. In some specific case, precedence constraints may be given a time window or time lags.

2.2 The Problem Space Search

The key to problem space is the utilization of a fast, problem-specific base heuristic. Problem space and its resultant search neighborhood is based on the fact that a heuristic is a mapping from a problem to a solution: H(P)=S. Thus a heuristic, problem pair (H,P) is an encoding of a solution sequence. Formally, the problem space optimization problem is defined as follows:

$$\text{Min}_{\delta \in D} \; V(H(P_0+\delta) \; ; \; P_0)$$

V(S;P): the objective function value of solution sequence S to problem(data) P

P_0 : an Nx1 vector representing original problem data

H : an heuristic which generates feasible solutions based on perturbation data

δ : an Nx1 perturbation vector, $P = P_0 + \delta$

D : the domain of perturbation vectors; a subset of $\Re^N$, the set of N dimensional real vectors.

This optimization problem is defined in the domain of the problem perturbation vector $\delta \in D$, or equivalently, in the space of "dummy problems" $P=P_0 + \delta$, thus the name "problem space". Note also that the original problem data vector P_0 is always used to evaluate the objective function.

To develop a local search procedure for this optimization problem, it is necessary to define a neighborhood structure. Since the domain of the problem is $\Re^N$ (or a subset of $\Re^N$), natural metrics (e.g., Euclidian distance) exist which, in turn, can be used to define a neighborhood.

A problem space neighborhood can be defined as follows:

$N(P_i)=\{P_j \mid P_j = T(P_i),$ where $P_j = P_0 + \delta_j$ and $\delta_j \in D\}$

N: the neighborhood function

T : a transformation of the data set, e.g., a sampler of neighboring problems or a neighborhood generation heuristic.

Storer et al. (1994) extended the problem space search using local search methods such as genetic algorithms, simulated annealing, and tabu search. Genetic algorithms were shown to be the most promising in the problem space. Storer et al. (1993) further characterized the properties of problem space and applied it to the number partitioning problem, JSP, placement problem for VLSI. They further investigated applications involving "robust scheduling" in an uncertain environment and showed encouraging results (Leon, Wu, and Storer, 1994; Storer and Wu, 1992).

Construction managers often need to consider arbitrary objective functions and constraints in construction project scheduling. Storer and Wu (1992) showed that a problem space neighborhood coupled with a genetic algorithm search method can be used to handle random objective functions. The advantage of problem space is the simplicity of using base heuristics which guarantees the generation of a feasible solution, and at the same time allows flexibility to handle arbitrary objective functions. In this research a stochastic local search is applied including genetic algorithm and evolutionary programming to RCPS. As chromosal representation a vector of perturbed processing times is used. A major advantage of this representation in genetic algorithms is that standard crossover operations can be applied without worrying about solution feasibility. Further, a directional search method is implemented using partial perturbation in evolutionary programming.

3. Base Heuristic for RCPS

A problem space local search method requires the application of a fast and simple base heuristic at each iteration of the search. Using a non-delay scheduling procedure, two base heuristics are tested for RCPS: Shortest processing time (SPT) and minimum slack (MINSLK) heuristics. The MINSLK heuristic prioritizes activities corresponding to their slack, which is the difference between latest finish time and earliest finish time in the CPM schedule. In general MINSLK has shown to be a good heuristic in RCPS (Davis and Patterson, 1975), although Schanmuganayagam (1989) proposed a "current float" method as an alternative to MINSLK for saving computation time. The current float (CF) indicates how much of an activity is behind schedule as follows:

$CF_i = LFT_i - TNOW - d_i$

LFT_i : latest finish time for activity i based on a CPM schedule.

This algorithm has been implemented and tested on Patterson's (1984) data set, and used as the base heuristic in problem space. The procedure, based on Schanmuganayagam (1989), is as follows:

Step 0. Define and initialize

AS: available set; the set of starting activities with all predecessors completed.

PS: process set; possible schedule or in-process set (=0).

FS: finished set; a set of processed activities (=0).

TNOW: current time (= 0).

Step 1. Compute latest finish time (LFT_i) for each activity i based on a CPM schedule.

Step 2. Compute CF_i for activities in AS, and

sort AS based on CF (ties are broken by a secondary priority rule).

Step 3. Select PS from the sorted AS based on resource constraints.

If the resource requirements are satisfied,

transfer AS to PS and assign each activity a starting time and

finishing time, and update the remaining resource status.

Set NEXTT=Min (finishing time in PS).

Step 4. Update TNOW as NEXTT , release resources from FS, and update AS.

Step 5. If FS does not include all activities, go to Step 2.

Since SPT (or MINSLK) dispatching is used to prioritize waiting jobs, an obvious problem space strategy is to perturb the processing times of operations. Specifically, if P(i) is a processing time for activity (i), the processing times of activity P(i) are perturbed by adding a uniform($-\theta,\theta$) random variable to yield "dummy" processing times DP(i). These dummy processing times are

used at the scheduling decision points in the application of the SPT (or MINSLK) dispatching rule to force the scheduling algorithm to produce different sequences. When computing the makespan, original problem data are used. For details, see Appendix A.

4. Rank Space - A Hybrid of Problem and Solution Space

A feasible scheduling solution can be represented as sequence or rank among activities. The rank can be then used as dummy (artificial) processing time for each activity. This rank-based artificial processing time facilitates a one-to-one mapping between the solution and the problem space. This mapping warrants a hybrid of problem and solution space search where an incumbent solution can be moved from the problem space to the solution space for a more thorough exploitation of local improvements. A main advantage of this hybrid is that problem space provides a parametric exploration of the search space while the solution space provides a refined exploitation of a specific region. In problem space one uses problem perturbation, in solution space one may use general local search techniques (e.g. k-swap with feasible solutions). Typically, when a solution is stuck at local optima, the local search method restarts from a new initial solution. But hybrid search uses problem space neighborhoods to guide further explorations of the search space. The hybrid method works very well in job shop scheduling (JSP).

A simple procedure can be proposed which generates rank space points (GRSP) corresponding to problem space points. For details, see Appendices C and D.

Step 1. For each problem space point P_i, compute its corresponding
solution space point, $S_i \leftarrow H(P_i)$.

Step 2. For each S_i, compute its corresponding rank space point, $R_i \leftarrow H^{-1}(S_i)$.

Step 3. Return vector R_i as the dummy processing time DP_i in problem space; modify the domain of the perturbation vector to proper scale.

5. Problem Space Search Method

5.1 Genetic Algorithms (GA's)

Genetic algorithms (Holland, 1975; Goldberg and Lingle, 1985; Goldberg, 1989) are probabilistic search methods conceived by analogy to nature and the process of evolution. They are parallel optimization algorithms in which a population of solutions is maintained and bred in successive generations in an attempt to discover improved solutions. Key to GA's is the manner in which solutions are "bred". Usually, GA's consist of three operators: selection, crossover, and mutation, along with a population of individual solutions. Parents are selected probabilistically with preference (fitness) given to better solutions. A crossover operator is used to produce offspring solutions from two parents. For sequencing problems, modified crossovers (partially mapped crossovers, order crossovers, and cycle crossovers) were investigated by various authors. GA's provide for both exploitation of the best solutions and exploration of the search space. The use of high mutation ratios may lead the algorithm to nearly a random search. Although GA's often exhibit very fast convergence to an approximate solution in a search space, a GA does not entail a mechanism for local fine tuning. Past applications of GA's use operation sequence as the chromosomal encoding. Here, the chromosomes are represented as dummy processing times. This encoding allows one to use the standard crossover operator and get feasible offspring solutions. In previous studies (Storer et al, 1992, 1993), applying a GA in problem space search showed consistently superior results. Therefore, in this research, a GA is implemented as the base algorithm (see GAPS in Appendix B). The procedure is as follows:

Step 0. Generate an initial population $P_i = P_0 +$ Uniform $(-\theta,\theta)$. Repeat for each generation.

Step 1. Compute $S_i = H(P_i)$ based on a non-delay schedule with SPT or CF or rank space.

Step 2. Asexual reproduction with mutation and power for selectivity of the fitness function. Sexual reproduction with crossover, mutation, and power for selectivity of the fitness function.

Return to Step 1.

End

5.2 Evolutionary Programming (EP)

Evolutionary programming (or evolution strategy) is a stochastic search method in function optimization and some control system (Fogel, 1991; Hoffmeister and Back, 1991; Schwefel, 1981). But in discrete optimization the application of EP has been limited. Generally the effect of mutation which is used as the main search operator in EP is the same as the "swap operation" in combinatorial optimization. EP generally works for simple "swap" kinds of problems (e.g., traveling salesman problem, graph partitioning, and module placement). In difficult sequencing problems a typical local search neighborhood (e.g., swapping neighborhood) could face major difficulty because of the need for procedures to check for solution feasibility.

The differences between GA's and EP are as follows. While the main operator in a GA is crossover, the main operator in EP is mutation. GA's work based on global tuning while EP works based on local fine tuning with self adaption. GA's work for binary or symbolic attributes; EP works for continuous variables. In EP, the search direction and step length are random, while a small number of variables are fixed (Schwefel, 1984). The basic procedure for EP is as follows. For details, see Appendix E.

Step 0. Generate an initial population. Mutate randomly using uniform $(-\Theta, \Theta)$

Step 1. Select parent (μ) from an initial population based on cost value.
Sort and select top μ from the initial population.

Step 2. For each generation, generate offspring (λ) by:
1) recombination selection $(1,2,..\mu)$ and
recombination with discrete and intermediate operator.
2) mutation with Normal $(0, \sigma^2)$

Step 3. Compute cost function and fitness function.

Step 4. Select (μ) for next generation from offspring.

End

One might view EP as a modified or pseudo steepest-descent-search method. One implemented extension of EP using problem space as a meta search space suggests a promising search direction through the partial perturbation of the problem data.

In the context of problem space, the idea of partial perturbation can be viewed as a directional search in a hyperplane which defines a "promising" search area. For example, perturb k of n elements on the problem data vector where the k elements can be selected either randomly or systematically. A regression analysis, for example, can be used to identify the most promising k-dimensional hyperplane based on a number of sample trials. This yields a k-dimensional basis for reducing a dimensional search space from $\Re^N$ to $\Re^k$. This reduction in dimension has potential for the development of an efficient search in problem space.

The idea of using insight from the problem itself may be used to identify promising search directions by modifying the neighborhood structure and/or the perturbation pattern. In JSP or RCPS, for example, we must resolve conflicts between jobs requesting the same resource at the same time. If processing times are encoded as a list of elements, it is possible, for example, to perturb just the operations (activities) which require the same machine at the same time. For example, one could perturb activities which lie on the critical path of the incumbent schedule.

6. Computational Experiments

Ten 51-node (activity) test problems were selected from Patterson (1984). These, in turn, were modified to create larger size (up to 408 activities) problems. This was accomplished by duplicating the 51 nodes up to 8 times and connecting them in parallel. The resulting problem resembles a multi-project scheduling problem. Because of the structure used, one expects significant competition for resources and, thus, challenging problems. The parameters for the base GA are set as follows:

$$\ominus = (1/2) * \text{max. processing time}$$

$$\text{population size} = 50$$

$$\text{mutation rate} = 15\ \%$$

$$\text{crossover rate} = 80\ \%$$

1000 instances (20 generations) were generated for a small-size problem and 2500 instances (50 generations) for a large-size problem. Results of the experiment appear in Table 1.

In EP the effect of partial perturbation from the above data set is demonstrated using the (μ, λ) method (μ=10, λ=100). The parents (μ) produce λ offsprings in each generation. Let θ=5 for initial population and σ=3 for interim populations. 1000 instances (10 generations) were generated for the small-size problem and 3000 instances (30 generations) for large-size problem. The results were compared with the known optimal solution for the small problem and showed the improvement based on heuristics itself for the large-size problem. Results of this experiment appear in Table 2. The effect of rank space is comparable to that of the SPT. The results of the MINSLK rule appear much better than that of SPT rule. The MINSLK results are consistently optimal solutions for small-size problems, and provide significant improvement (from heuristic solutions) for large-size problems. There is no significant difference between the results of EP and those of GA's. However, the results of GA's converged earlier for the same performance.

7. Conclusions

In this paper, problem space is applied to the resource constrained project scheduling problem (RCPS). Extensions of genetic algorithms (GA's) and evolutionary programming (EP) are shown using "problem space". This same method can be applied to any (or multi) objective in RCPS. This research suggests useful tools for project managers who seek to optimize planning, scheduling, and control of construction projects.

Table 1. Results of SPT, RANK, and MINSLK using Genetic Algorithms (GA's)

Problem	Node	SPT			RANK		MINSLK			
		init	fin1	imp1	fin2	imp2	init	fin3	imp3	OPT
101	51	82	78	4.9	75	8.5	78	75	3.8	75
102	51	101	83	17.8	83	17.8	83	83	0	83
103	51	62	56	9.7	56	9.7	56	56	0	56
104	51	87	79	9.2	80	8.7	80	79	1.2	79
105	51	85	77	9.4	77	9.4	77	76	1.3	76
106	51	75	60	20	61	18.6	63	60	4.7	60
107	51	84	79	5.9	78	7.1	78	78	0	78
108	51	76	61	19.7	63	17.1	65	61	6.1	61
109	51	73	61	16.4	61	16.4	63	60	4.7	60
110	51	54	51	5.5	51	5.5	52	51	1.9	50
a101a	102	148	134	9.5	140	5.7	152	131	13.8	unkn
b108a	102	124	106	14.5	108	12.9	117	103	11.9	unkn
c109a	102	124	106	14.5	109	12.1	114	106	7.0	unkn
a101b	153	211	199	5.7	201	4.7	225	194	13.8	unkn
a108b	153	172	153	11	156	9.3	172	151	12.2	unkn
c109b	153	187	163	12.8	160	14.4	165	157	4.8	unkn
a101c	204	278	266	4.3	269	3.2	304	259	14.8	unkn
b108c	204	239	220	7.9	225	5.8	236	216	8.4	unkn
c109c	204	243	220	9.5	214	11.9	224	207	7.5	unkn
a101c2	408	541	525	3	524	3.1	598	507	15.2	unkn
b108c2	408	486	437	10	441	9.3	484	426	12	unkn
c109c2	408	473	432	8.6	430	9.5	446	412	7.6	unkn

* SPT: shortest processing time first rule; RANK: rank space rule
MINSLK: minimum slack first rule; init:initial solution based on heuristic itself
fin1: final result using SPT; fin2: final result using RANK
fin3: final result using MINSLK; imp= % improvement, based on heuristic
OPT=optimal solution, unkn=unknown

Table 2. Results of MINSLK based on partial and full perturbation from Evolutionary Programming (EP)

Problem	Node	init	fin1	fin2	imp1	imp2	OPT
101	51	78	75	75	3.8	3.8	75
102	51	83	83	83	0	0	83
103	51	56	56	56	0	0	56
104	51	80	79	79	1.2	1.2	79
105	51	77	76	76	1.3	1.3	76
106	51	63	60	60	4.7	4.7	60
107	51	78	78	78	0	0	78
108	51	65	61	61	6.1	6.1	61
109	51	63	60	60	4.7	4.7	60
110	51	52	51	51	1.9	1.9	50
a101a	102	152	132	132	13.1	13.1	unkn
b108a	102	117	104	104	11.1	11.1	unkn
c109a	102	114	106	106	7	7	unkn
a101b	153	225	196	195	12.8	13.3	unkn
a108b	153	172	153	151	11	12.2	unkn
c109b	153	165	157	156	4.8	5.4	unkn
a101c	204	304	260	257	14.5	15.5	unkn
b108c	204	236	217	215	8	8.9	unkn
c109c	204	224	209	206	6.7	8.0	unkn
a101c2	408	598	509	507	14.8	15.2	unkn
b108c2	408	484	442	439	8.7	9.2	unkn
c109c2	408	446	413	412	7.4	7.6	unkn

* MINSLK: minimum slack first rule
init = initial solution based on heuristic itself
fin1 = final result from perturbing all elements using MINSLK
fin2 = final result from perturbing 20% of all elements using MINSLK
imp = % improvement, based on heuristic
OPT = optimal solution, unkn = unknown

References

1. Bell, C.E. and Park, K., (1990). "Solving Resource-Constrained Project Scheduling Problems by A* Search", *Naval Research Logistics*, Vol 37, pp. 61-84.
2. Boctor, F.F., (1990). "Some Efficient Multi-Heuristic Procedures for Resource Constrained Project Scheduling", *European Journal of Operational Research,* 49, pp. 3-13.
3. Christofides, N., Alvarez-Vades, R., and Tamarit, J.M., (1987). "Project Scheduling with Resource Constraints: A Branch and Bound Approach", *European Journal of Operational Research,* 29, pp. 262-273.
4. Chrzanowski, E. N. and Johnston, D. W. (1986) "Application of Linear Scheduling", *J. Constr. Engrg. and Mgmt.*, ASME, 112 (4), pp. 476-491.
5. Davis, E. W. and Patterson, J.H.,(1975). "A Comparison of Heuristic and Optimal Solutions in Resource-Constrained Project Scheduling", *Management Science*, 21 (8), pp. 944-955.
6. Goldberg, D.E., and Lingle, R. (1985) "Alleles, Loci, and the Traveling Salesman Problem", in J. Greffenstette (ed.) *Proceedings, International Conference on Genetic Algorithms and Their Applications;* Carnegie-Mellon University, Pittsburgh, PA.
7. Leon, V. J., Wu, S. D., and Storer, R. H. (1994) "Robustness Measures and Robust Scheduling for Job Shop" *IIE Trans*actions, in print.
8. Li, K. Y., and Willis, R. J., (1992) "An Iterative Scheduling Technique for Resource-Constrained Project Scheduling", *European Journal of Operational Research* 56, pp.370-379, North-Holland Publishers.
9. Fogel, D. B. (1991) *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*, Ginn Press.
10. Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison-Wesley.
11. Hoffmeister, F. and Back, T. (1991), "Genetic Algorithms and Evolution Strategies - Similarities and Difference", *Parallel Problem Solving from Nature,* Springer-Verlag, pp. 455-469.

12. Holland, J. H. (1975) *Adaption in Natural and Artificial Systems* University of Michigan Press, Ann Arbor.

13. Jaafari, A. (1984) "A Criticism of CPM for Project Planning" *J. Constr. Engrg. and Mgmt.*, ASME, 110 (2), pp. 316-334.

14. Olaguibel, A. V., Goerlich, J.M. (1989) "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and Empirical Analysis" *Advances in Project Scheduling,* Elsevier Science Publishers B.V., pp. 113-134.

15. Patterson, J.H. (1984) "A Comparison of Exact Approaches for Solving the Multiple Constrained Project Scheduling", *Management Science,* 30 (7), pp. 854-867.

16. Pultar, M. (1990) "Progress-Based Construction Scheduling", *Journal of Construction Engineering and Management,* ASME, Vol. 116, No. 4, pp. 670-688.

17. Scherer, W.T. and Rotman, F. (1992) "Combinatorial Optimization for Spacecraft Scheduling", *Proceedings 1992 IEEE International Conf. on Tools with AI,* Arlington, VA, Nov. pp.120-127.

18. Schwefel, H.P. (1981) *Numerical Optimization of Computer Models*, John Wiley & Sons, Ltd.

19. Shanmuganayagum, V. (1989). "Current Float Technique for Resources Scheduling", *Journal of Construction Engineering and Management,* Vol. 115, no. 3, Sept., pp. 401-411.

20. Stinson, J.P., Davis, E.W., and Khumawala, B. M., (1978). "Multiple Resource Constrained Scheduling Using Branch & Bound", *AIIE Transactions*, 10 (3), pp. 252-259.

21. Storer, R.H., Wu, S.D., and Vaccari, R. (1992) "New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling", *Management Science,* vol. 38, no.10, Oct., pp. 1495-1509.

22. Storer, R.H., Wu, S.D. and Vaccari, R. (1994). "Local Search in Problem and Heuristic Space for Job Shop Scheduling", ORSA Journal on Computing, in print.

23. Storer, R. H. and Wu, S. D. (1992) "Job Scheduling with Random Objective Functions", *1992 First Industrial Engineering Research Conference Proceedings*, pp.401-404.

24. Storer, R.H., Wu, S.D. and Park, I. (1993). "Genetic Algorithms in Problem Space for Sequencing Problems", *Operations Research in Production Planning and Control*, Springer-Verlag , pp. 584-597.

25. Tavares, L.V. and Weglarz, J. (1990). "Project Management and Scheduling: A Permanent Challenge for OR", *European Journal of Operational Research*, 49, pp. 1-2.

26. Talbot, F. B. and Patterson, J. H. (1978) "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource Constrained Scheduling Problems", *Management Science,* July, Vol.24, no. 11, pp. 1163-1174.

27. The Construction Industry Institute Strategic Planning Committee (1990) "Assessment of Construction Industry Project Management Practices and Performance" A Special CII Publication.

28. Tsubakitani, S. and Deckro, R.F. (1990) "A Heuristic for Multi-Project Scheduling with Limited Resources in the Housing Industry", *European Journal of Operational Research,* 49, pp. 80-81.

Appendix A. Procedure for Generation Schedule Based on Heuristics ($H_{heuristic}$)

Step 0. Define and initialize

Heuristic: dispatching rule (e.g. SPT or MINSLK)

AS: available set; the set of starting activities with all their predecessors completed.

PS: process set, possible schedule or in-process set (=0)

FS: finished set, a set of processed activities (=0)

TNOW: current time (= 0)

Step 1. If Heuristic = SPT, then go to Step 2.

Compute latest finish time (LFTi) for each activity i based on CPM schedule.

Step 2. Compute current float, $CF_i = LFT_i - TNOW - d_i$

for activities in AS.

If Heuristic = MINSLK, Sort AS based on CF.

If Heuristic = SPT, Sort AS based on d_i.

(Ties are broken by a secondary priority rule).

Step 3. Select PS from the sorted AS based on resource constraints

If the resource requirements are satisfied,

transfer AS to PS and assign each activity a starting time and finishing time, and update the remaining resource status

Set NEXTT=Min (finishing time in PS)

Step 4. Update TNOW as NEXTT , release resources from FS, and update AS.

Step 5. If FS does not include all activities, Go to Step 2.

* * * *

Appendix B. Algorithm GAPS

Step 0. Create the initial population.

Set g=0. Set Maxmake=0. Set BestMakespan=∞.

For j=1 to Population Size (Pop Size) D_0

Generate a perturbation vector D_j

Create dummy processing times $DP_{0,j} = P_0 + D_j$

Apply $H_{heuristic}$ $(DP_{0,j}, P_0)$ to obtain makespan $M_{0,j}$.

If ($M_{0,j}$ > MaxMake), Set MaxMake=$M_{0,j}$.

If ($M_{0,j}$ < BestMakespan), Set BestMakespan=$M_{0,j}$

Go to next j

Step 1. For Generation g=1 to NumGen D_0

a) Evaluate fitness of initial population

SumFit=0

For j=1 to PopSize D_0

SumFit = SumFit + [MaxMake - $M_{g-1,j}$]$^{\pi}$

Go to next j

For j=1 to PopSize D_0

FIT j = [MaxMake - $M_{g-1,j}$]$^{\pi}$ / SumFit

Go to next j

b) Do Asexual Reproduction

For j=1 to INT(PopSize*AsexRate)

(select a number of the population according to FIT:)

i=FunctionSelect(FIT):

Set $DP_{g,j} = DP_{g-1,i}$

Step 1. continued

c) Do Sexual Reproduction

For j=INT(PopSize*Asexrate) + 1 to PopSize,

Select a Mother m and Father f according to FIT:

m=FunctionSelect(FIT):

f= FunctionSelect(FIT):

Set CrossSite = INT(Random*StringLength +1)

Set $DP_{g,j}$ = FunctionCrossOver($DP_{g-1,m}$, $DP_{g-1,f}$, CrossSite)

Go to next j

d) Evaluate Makespans of the new generation

Set MaxMake=0.

For j=1 to PopSize D_0

Apply $H_{heuristic}$ ($DP_{g,j}$, P_0) to obtain makespan $M_{g,j}$.

If ($M_{g,j}$ > MaxMake), Set MaxMake=$M_{g,j}$.

If ($M_{g,j}$< BestMakespan), Set BestMakespan=$M_{g,j}$.

Go to next j

Go to next g

Step 2. Report BestMakespan

* * * *

Appendix C. Algorithm GA-RANK

Step 0. Create the initial population

Set g=0. Set Maxmake=0. Set BestMakespan=∞.

For j=1 to PopSize D_0

Generate a perturbation vector D_j

Create dummy processing times $DP_{0,j} = P_0 + D_j$

Apply H_{spt} ($DP_{0,j}$, P_0) to obtain makespan $M_{0,j}$

Get artificial dummy processing time $DP_{0,j} = R_{0,j} = (H^{-1}_{spt} (S_j))$

If ($M_{0,j}$ > MaxMake), Set MaxMake=$M_{0,j}$.

If ($M_{0,j}$< BestMakespan), Set BestMakespan=$M_{0,j}$

Go to Next j

Step 1. For Generation g=1 to NumGen D_0

a) Evaluate fitness of initial population

SumFit=0

For j=1 to PopSize D_0

SumFit = SumFit + [MaxMake-$M_{g-1,j}$]

Go to Next j

For j=1 to PopSize D_0

FIT j = [maxmake - $M_{g-1,j}$] / SumFit

Go to Next j

b) Do Asexual Reproduction

For j=1 to INT(PopSize*AsexRate)

(select a number of the population according to FIT:)

i=FunctionSelect(FIT):

Set $DP_{g,j} = DP_{g-1,i}$

c) Do Sexual Reproduction

For j=INT(PopSize*Asexrate) + 1 to PopSize

(select a Mother m and Father f according to FIT:)

m=FunctionSelect(FIT):

f= FunctionSelect(FIT):

Set CrossSite = INT(Rand*StringLength +1)

Set $DP_{g,j}$ = FunctionCrossOver($DP_{g-1,m}$, $DP_{g-1,f}$, CrossSite)

Go to Next j

d) Evaluate Makespans of the new generation

Set MaxMake=0.

For j=1 to PopSize D_0

Apply H_{spt} ($DP_{g,j}$, P_0) to obtain makespan $M_{g,j}$.

If ($M_{g,j}$ > MaxMake), Set MaxMake=$M_{g,j}$.

If ($M_{g,j}$ < BestMakespan), Set BestMakespan=$M_{g,j}$.

Go to Next j

Go to Next g

2. Report BestMakespan

* * * *

Appendix D. Procedure RANK space

Let j be the index of jobs (j=1,J)

Let m be the index of machines (m=1,M)

Let ma(m) be index of machine m (ma(m)=1)

Let n be the index of operations (n=1,N where N=JxM)

Let P_0 be the Nx1 vector of processing times

Let D_{ij} be an Nx1 vector with each element randomly generated

as Uniform(-θ ,θ) (i=1, Neighborhood size ; j=1,N)

Let $H(P_0 + D_i, P_0)$ be the hybrid dispatching rule heuristic which calculates priorities using $P_0 + D_i$ and generates the schedule and makespan using P_0.

Where θ = (1/2)* maximum processing time, x=(1/2)* 0J

Step 0. Generate an initial population:

Do i=1, Neighborhood size

$P_i = P_0$ + Uniform(-θ,θ)

Continue

Step 1. Generate schedule and dummy processing times as follows:

Do i=1, Neighborhood size

GEN: call schedule (nondelay or active) with SPT dispatching rule.

if operation n scheduled at machine m then

assign dummy processing times $D_{i,n}$ = ma(m) at operation n:

ma(m)=ma(m)+1:

increment scheduled operation :

if all operations are scheduled, go to GO1:

else go to GEN:

GO1: clear machine index: ma(m)=1, m=1, M.

Continue

Step 2. Compute cost function using $H(P_0 + D_i, P_0)$

Step 3. Generate interim population from incumbent solution:

Do i = 1, Neighborhood size

perturb $D_{i,n}$ = $D_{incumbent,n}$ + Uniform(x,x)

Continue

Go to Step 1.

* * * *

Appendix E. Algorithm EP-MINSLK

Step 0. Create the initial population

Set g=0. Set Maxmake=0. Set BestMakespan=∞. PopSize= $\mu*\lambda$:

For j=1 to PopSize D_0

Generate a perturbation vector D_j from Uniform $(-\theta,\theta)$

Create dummy processing times $DP_{0,j} = P_0 + D_j$

Apply H_{minslk} $(DP_{0,j}, P_0)$ to obtain makespan $M_{0,j}$

If $(M_{0,j} > \text{MaxMake})$, Set MaxMake=$M_{0,j}$.

If $(M_{0,j} < \text{BestMakespan})$, Set BestMakespan=$M_{0,j}$.

Go to Next j

Step 1. For Generation g=1 to NumGen D_0

a) Sort $M_{g-1,j}$ and select top μ based on the sorted Makespan $M_{g-1,j}$

b) Do Asexual Reproduction

For i=1 to μ

For k=1 to λ

j=(i-1)*λ + k

Set $DP_{g,j} = DP_{g-1,i}$ + Normal(0, σ^2)

Go to Next k

Go to Next i

d) Evaluate Makespans of the new generation

Set MaxMake=0.

For j=1 to PopSize D_0

Apply H_{minslk} $(DP_{g,j}, P_0)$ to obtain makespan $M_{g,j}$.

If $(M_{g,j} > \text{MaxMake})$, Set MaxMake=$M_{g,j}$.

If $(M_{g,j} < \text{BestMakespan})$, Set BestMakespan=$M_{g,j}$.

Go to Next j

c) Go to Next g

Step 2. Report BestMakespan

* * * *